

# Compression Complexity

Stephen Fenner

University of South Carolina

Lance Fortnow

Georgia Institute of Technology

February 15, 2017

## Abstract

The Kolmogorov complexity of  $x$ , denoted  $C(x)$ , is the length of the shortest program that generates  $x$ . For such a simple definition, Kolmogorov complexity has a rich and deep theory, as well as applications to a wide variety of topics including learning theory, complexity lower bounds and SAT algorithms.

Kolmogorov complexity typically focuses on decompression, going from the compressed program to the original string. This paper develops a dual notion of compression, the mapping from a string to its compressed version. Typical lossless compression algorithms such as Lempel-Ziv or Huffman Encoding always produce a string that will decompress to the original. We define a general compression concept based on this observation.

For every  $m$ , we exhibit a single compression algorithm  $q$  of length about  $m$  which for  $n$  and strings  $x$  of length  $n \geq m$ , the output of  $q$  will have length within  $n - m + O(1)$  bits of  $C(x)$ . We also show this bound is tight in a strong way, for every  $n \geq m$  there is an  $x$  of length  $n$  with  $C(x) \approx m$  such that no compression program of size slightly less than  $m$  can compress  $x$  at all.

We also consider a polynomial time-bounded version of compression complexity and show that similar results for this version would rule out cryptographic one-way functions.

# 1 Introduction

Kolmogorov complexity has a rich history, with many applications to areas such as computability, machine learning, number theory, and computational complexity. The book of Li & Vitányi [LV08] gives an excellent background on the subject.

Kolmogorov complexity measures the information content inherent in a string  $x$  by the length of the shortest program that computably produces  $x$ ; this length is denoted  $C(x)$ . We think of the program  $p$  as a compressed version of  $x$  and the algorithm producing  $x$  a decompression procedure. Kolmogorov complexity focuses on decompression procedures. In this paper we turn our attention to the task of producing  $p$  given  $x$ , i.e., compression.

How hard is it to compute the shortest program  $p$  from  $x$ ? It's equivalent to the halting problem since even the set  $R$  of random strings ( $x$  for which  $C(x) \geq |x|$ ) is hard for the halting problem [Kol65, Cha66, Sol64a, Sol64b].

If we are given the value  $C(x)$  then we can compute a program  $p$  of that length by dovetailing through all programs of that length until we find one that produces  $x$ . While this method works on a single string, it does not work in general and won't even halt if we are given an underestimate of the length.

Consider the lossless compression algorithms in use today. Among the general-purpose compression algorithms are Huffman coding, run-length encoding, Burrows-Wheeler, and Lempel-Ziv (and variants thereof) [Wik17]. All of these algorithms have a common property, for any input  $x$  they will produce an output that will decompress back to  $x$ .

To this end we define compression functions as having this single property. A *compression function* is a computable function  $q$  such that  $U(q(x)) = x$  for all strings  $x$ , for some fixed universal Turing machine  $U$ . The trivial compression function  $q(x)$  simply outputs the program "Print  $x$ ".

A compression algorithm may benefit from having information embedded in it; for example, a Huffman encoding tree (of letters, digrams, or even trigrams) to Huffman compress English text. Therefore, the size of a compression algorithm itself may improve its performance—at least on some strings.

We investigate a basic trade-off between the size of a general-purpose compression algorithm and how well it compresses.

Our main result shows roughly that for any value of  $m$ , there is a compression function  $q$  of size roughly  $m$  that will fully compress any string of Kolmogorov complexity at most  $m$ . We need only  $m$  bits for  $q$  even though

there are exponentially many (in  $m$ ) strings that  $q$  can fully compress.

We also show an essentially tight strong inverse: roughly, that for any  $n$  and  $m$  with  $m < n$  there is a single string  $x$  of length  $n$  and Kolmogorov complexity  $m$  such that every compression function  $q$  of size less than  $m$  fails to compress  $x$  at all.

We further consider polynomial-time bounded versions of compression complexity whose results depend on unproven assumptions in computational complexity. If  $\mathbf{P} = \mathbf{NP}$ , then one can have perfect efficient polynomial-time compression functions; on the other hand, even a small compression would break one-way functions.

Finally we explore some future directions for research into compression complexity.

## 1.1 Main Results

Our first theorem says that for any  $m$  there is a compression function of size about  $m$  that optimally compresses all strings  $x$  with  $C(x) \leq m$  and does not significantly expand other strings.

In what follows,  $k$  is a sufficiently large fixed constant independent of  $m$  or  $x$ . See the next section for detailed definitions.

**Theorem 1** *For every  $m$ , there exists a compression function  $q$  with  $|q| \leq m + k$  such that, for all strings  $x$ ,*

1.  $|q(x)| = C(x)$  if  $C(x) \leq m$ , and
2.  $|q(x)| \leq |x| + k$  otherwise.

**Corollary 2** *For every  $m$ , there exists a compression function  $q$  with  $|q| \leq m + k$  such that, for all  $x$  with  $|x| \geq m$ ,*

$$|q(x)| - C(x) \leq |x| - m + k.$$

The next theorem implies that Theorem 1 and its corollary are essentially tight. It says for any  $n \geq m$  that if a compression function  $q$  for length  $n$  (i.e., one that only needs to work on strings of length  $n$ ) is significantly shorter than  $m$ , then it behaves poorly on at least one string: there is an  $x$  of length  $n$ —independent of  $q$ —such that  $C(x)$  is about  $m$  but  $q$  does not compress  $x$  at all.

**Theorem 3** For all  $m, n$  with  $0 \leq m \leq n$  there exists an  $x$  such that

1.  $|x| = n$ ,
2.  $C(x) \leq m + k \log n$ ,
3. For all compression functions  $q$  for length  $n$ , with  $|q| \leq m - k \log n$ ,  $|q(x)| \geq n$ , and
4. For all compression functions  $q$  for length  $n$ , with  $|q| \leq m - k \log n$ ,  $|q(x)| - C(x) \geq n - m - k \log n$ .

Note that (4) follows from (2) and (3). Compare (4) with the result of Corollary 2.

We prove Theorems 1 and 3 in Section 3. In Section 4 we give definitions and results for time-bounded compression.

## 2 Preliminaries

Our notation and basic definitions are standard. Here we briefly outline the basic definitions of Kolmogorov complexity. For an in-depth treatment, see the standard textbook on the subject by Li & Vitányi [LV08].

We assume all strings are over the binary alphabet  $\{0, 1\}$ . We define a standard pairing function  $\langle \cdot, \cdot \rangle$  injectively mapping pairs of strings to strings as

$$\langle x, y \rangle := 1^{|x|}0xy$$

for all strings  $x$  and  $y$ .

Fix a universal machine  $U$  suitable for defining Kolmogorov complexity. It suffices to choose  $U$  such that, for any Turing machine  $M$ , there exists a string  $p$  such that  $M(x) = U(\langle p, x \rangle)$  for any string  $x$  such that  $M(x)$  is defined. Abusing notation, we also write  $p(x)$  for  $U(\langle p, x \rangle)$  and call  $p$  a “program.”

Define  $C(x)$  as the length of the shortest  $p$  such that  $U(p) = x$ .

Both our main results freely refer to a constant  $k$ . We can (and do) choose  $k$  large enough (depending only on  $U$ ) such that our results hold.

**Definition 4** Let  $n$  be a natural number. A *compression function for length  $n$*  is a program  $q$  such that  $U(q(z)) = z$  for all strings  $z$  of length  $n$ . A *compression function* is a program  $q$  that is a compression function for all lengths.

Note that any compression function  $q$  is total, and for all  $x$ ,  $q(x) \geq C(x)$ .

Let  $\mathbf{BB}(m)$  (“Busy Beaver of  $m$ ”) be the maximum time for  $U(p)$  to halt over all halting programs  $p$  of length at most  $m$ . Let  $p_m$  be the lexicographically least program of length at most  $m$  that achieves  $\mathbf{BB}(m)$  running time.

### 3 Proof of the Main Results

In this section we give proofs of Theorems 1 and 3.

**Proof of Theorem 1:**

Given  $m$ , we define  $q(z)$  as follows:

- Let  $t$  be the number of steps used by  $U(p_m)$  before halting. Note that  $t = \mathbf{BB}(m)$ .
- Look for the lexicographically shortest program  $p$  of length at most  $|z|$  such that  $U(p) = z$  within  $t$  steps.
  - If  $p$  is found, then output  $p$ .
  - Otherwise, output “Print  $z$ .”

We hardwire the value of  $p_m$  into the code for  $q$  (e.g., let  $q = \langle r, p_m \rangle$  for some program  $r$  independent of  $m$ ), so  $|q| \leq m + k$ .

Fix a string  $x$ .

**Case 1:**  $C(x) \leq m$ .

Let  $p$  be the lexicographically first program of length  $C(x)$  such that  $U(p) = x$ . By the definition of  $\mathbf{BB}(m)$ ,  $U(p)$  will halt in at most  $t$  steps. So we’ll have  $q(x) = p$  and  $|q(x)| = C(x)$ .

**Case 2:**  $C(x) > m$ .

Either  $q(x)$  outputs “Print  $x$ ” or a program of length at most  $n$ . Either way  $|q(x)| \leq n + k$ . □

**Proof of Theorem 3:**

Let  $A_s^\ell$  be the set of strings  $y$  of length  $\ell$  such that there is no program  $p$  with  $|p| < \ell$  such that  $U(p)$  outputs  $y$  within  $s$  steps. One can compute a canonical index for  $A_s^\ell$  given  $s$  and  $\ell$  as input.  $A_s^\ell$  contains the random strings of length  $\ell$ , so in particular  $A_s^\ell$  is never empty. If  $s \geq \mathbf{BB}(\ell)$ , then  $A_s^\ell$  is exactly the set of random strings of length  $\ell$ .

Given  $m \leq n$ , let  $t = \mathbf{BB}(m)$  and  $x$  be the lexicographically first string in  $A_t^n$ .

Note that  $C(x) \leq m + k \log n$ , since we can describe  $x$  by  $p_m$ ,  $m$ , and  $n$ , using  $p_m$  to find  $t$ .

Suppose there is a compression function  $q$  for length  $n$  with  $|q| \leq m - k \log n$  such that  $|q(x)| < n$ .

Let  $z$  be the lexicographically first random string of length  $m$ . We show how to use  $q$ ,  $n$ , and  $m$  to find  $z$ . This will yield a contradiction to the fact that  $z$  is random.

Let  $t'$  be the maximum over all  $y$ ,  $|y| = n$ , of the number of steps required for  $U$  to halt on input  $q(y)$ . Here we use the fact that  $U(q(y)) = y$  for all  $y$  of length  $n$ .

Since  $|q(x)| < n$  and  $U(q(x)) = x$ , by the definition of  $x$  the number of steps  $\hat{t}$  required for  $U$  to halt on input  $q(x)$  must be greater than  $t = \mathbf{BB}(m)$ . So we have  $t' \geq \hat{t} > t = \mathbf{BB}(m)$ . We can't necessarily compute  $t$  or  $\hat{t}$  from just  $q$ ,  $n$ , and  $m$ , but we can compute  $t'$ .

Now compute  $A_{t'}^m$ , which will be exactly the random strings of length  $m$ , and  $z$  will be the lexicographically least string in that set.

□

## 4 Time-Bounded Compression Complexity

The proofs in Section 3 create machines that run in time based on the busy-beaver function, which grows faster than any computable function. Practical compression and decompression algorithms need to be far more efficient. We explore a polynomial time-bounded version of compression complexity in this section, though, not too surprisingly, the results we get will depend on open questions in computational complexity.

Time bounds can play into both the compression and decompression procedures.

**Definition 5** An  $(f, g)$ -time bounded compression function is a function  $q$  such that for all strings  $x$ ,

1.  $q(x)$  halts within  $f(|x|)$  steps.
2.  $U(q(x))$  halts and outputs  $x$  within  $g(|x|)$  steps.

Let  $C^t(x)$  be the length of the shortest program  $p$  such that  $U(p)$  outputs  $x$  within  $t(|x|)$  steps. For every  $(f, g)$ -compression function  $q$ ,  $|q(x)| \geq C^g(x) \geq C(x)$ .

Typically we consider  $f$  and  $g$  as polynomials. For a fixed polynomial  $p$  we can easily compute the smallest  $p$ -time bounded program for  $x$  in  $\mathbf{FP}^{\mathbf{NP}}$ . As a corollary we get

**Proposition 6** *If  $\mathbf{P} = \mathbf{NP}$  then for every polynomial  $p$  there is a polynomial  $p'$  and a  $(p', p)$ -compression function  $q$  such that  $|q(x)| = C^p(x)$ .*

So in particular, to show that we don't have perfect efficient compression would require settling the  $\mathbf{P}$  versus  $\mathbf{NP}$  problem.

On the other hand, we show that if we can have short compression programs that more than trivially compress all strings that have significant compressions, à la Theorem 1 and Corollary 2, then we can break one-way functions.

First we need to define a family of compression functions.

**Definition 7** *A family of  $(f, g)$ -time bounded compression functions is an enumeration of programs  $q_0, q_1, \dots$  such that for all  $n$  and all strings  $x$  of length  $n$ ,*

1.  $q_n(x)$  halts within  $f(n)$  steps.
2.  $U(q_n(x))$  halts and outputs  $x$  within  $g(n)$  steps.

We say a compression family has *polynomial-size* if for some constant  $c$ ,  $|q_n| \leq n^c$  for all  $n \geq 2$ . A compression family is *uniform* if there is a polynomial-time algorithm  $Q$  such that  $Q(1^n) = q_n$  for all  $n$ .

**Theorem 8** *Fix a constant  $\delta$  with  $0 < \delta < 1$ . Suppose that for any polynomial  $p'$  there is a polynomial  $p$  and a polynomial-size family of  $(p, p')$ -compression functions  $q_0, q_1, \dots$  such that for all  $n$  and  $x \in \Sigma^n$  with  $C^{p'}(x) \leq n^\delta$ , we have  $|q_n(x)| < n$ . Then one-way functions do not exist relative to polynomial-size circuits.*

*In particular, we could factor numbers on average with polynomial-size circuits. Under the additional assumption that the family is uniform we can factor numbers on average with a polynomial-time algorithm.*

**Proof:** Let's assume we have a one-way function  $f$ . Håstad, Impagliazzo, Levin and Luby [HILL99] show how to convert this one-way function into a polynomial-time pseudorandom generator  $G : \Sigma^{n^\epsilon} \rightarrow \Sigma^n$  (for any fixed  $\epsilon > 0$ ) so that no polynomial-size circuit can distinguish the output of  $G$  on a random seed from a truly uniformly chosen string of length  $n$ .

Choose an  $\epsilon$  so that  $0 < \epsilon < \delta$ . Pick  $p'(n)$  larger than the running time of  $G$  and let  $q_0, q_1, \dots$  be the family of  $(p, p')$ -compression functions given in the assumptions of Theorem 8.

Consider the following test  $T(x)$  that can be expressed as a polynomial-sized circuit: Output 0 if  $|q_n(x)| < n$ , and output 1 otherwise.

Buhrman, Jiang, Li and Vitányi [BJLV00] show that a constant fraction of the strings of length  $n$  have  $C(x) \geq n$ . Since  $|q_n(x)| \geq C(x)$ , if we choose a string  $x$  at random, then  $T(x)$  will output 1 with probability at least some constant  $\alpha > 0$ .

Suppose  $x$  is the output of  $G(r)$  for some  $r$ . We can describe  $x$  by the code for  $G$  and  $r$ , or  $n^\epsilon + O(1) \leq n^\delta$  bits for sufficiently large  $n$ . Since the running time of  $G$  is less than  $p'$ , we have  $|C^{p'}(x)| \leq n^\delta$ . By the assumptions of Theorem 8 we have  $|q(x)| < n$  and  $T(x) = 0$ .

$T(x)$  will output 1 with probability at least  $\alpha$  when  $x$  is chosen at random and with probability 0 when  $x$  is the output of  $G$  on a randomly chosen seed. This contradicts the fact that  $G$  is a pseudorandom generator and  $f$  is a one-way function.

In particular, the function that maps two primes to their product is not one-way, so we can factor randomly chosen numbers using polynomial-size circuits.

If the family of compression functions is uniform, then  $T(x)$  above can be expressed as a polynomial-time algorithm. Håstad, Impagliazzo, Levin and Luby [HILL99] show how to take any one-way function against polynomial-time algorithms and convert it to a pseudorandom generator that has no uniform tests. Putting this together there must be a polynomial-time algorithm that factors randomly chosen numbers.  $\square$

## 5 Future Directions

This work is just the start of compression complexity.

Is there a compression analogue of conditional Kolmogorov complexity  $C(x|y)$ ? The results of Section 1.1 should go through if we allow  $q$  to have

access to  $y$ , but it is less clear what happens if  $q$  does not have access to  $y$ .

Symmetry of Information shows that for any strings  $x$  and  $y$ ,  $C(x, y) \leq C(x) + C(y|x) + O(\log(|x| + |y|))$ . Suppose we consider compression functions  $q(x, y)$  to produce programs  $p_1$  and  $p_2$  such that  $U(p_1) = x$  and  $U(\langle p_2, x \rangle) = y$ . How short can we get  $|p_1| + |p_2|$  compared to  $C(x, y)$ ?

There are several other notions of time-bounded Kolmogorov complexity (see [LV08]) such as distinguishing complexity, where we need only distinguish a string  $x$  from other strings. Do the results of Section 4 still apply?

One could also consider lossy compression, perhaps building on Kolmogorov complexity with errors [FLV06].

## Acknowledgments

We would like to thank Eric Allender, Saurabh Sawlani, and Jason Teutsch for helpful discussions.

## References

- [BJLV00] Harry Buhrman, Tao Jiang, Ming Li, and Paul Vitányi. New applications of the incompressibility method: Part II. *Theoretical Computer Science*, 235(1):59 – 70, 2000.
- [Cha66] Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, October 1966.
- [FLV06] Lance Fortnow, Troy Lee, and Nikolai Vereshchagin. *Kolmogorov Complexity with Error*, pages 137–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
- [Kol65] Andrei Kolmogorov. Three approaches for defining the concept of information quantity. *Problems of Information Transmission*, 1:1–7, 1965.
- [LV08] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. New York, third edition, 2008.

- [Sol64a] R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7(1):1–22, 1964.
- [Sol64b] R.J. Solomonoff. A formal theory of inductive inference. Part II. *Information and Control*, 7(2):224–254, 1964.
- [Wik17] Wikipedia. Lossless compression — Wikipedia, the free encyclopedia, 2017. [Online; accessed 09-February-2017].